DS2 : Déduction naturelle et langages

Consignes
Vous traiterez un et un seul des deux sujets suivants :
- DS2 : Il est constitué des problèmes 1 et 3.
- DS2* : Il est constitué des problèmes 1 et 2.

Vous indiquerez **clairement** le sujet traité sur votre première copie. Dans les deux cas, les deux problèmes sont indépendants et peuvent être traités dans un ordre quelconque. Le problème commun comptera pour environ 1/3 des points et le problème spécifique pour 2/3. Les documents et la calculatrice sont interdits.

Consignes spécifiques pour le DS2*
Le problème 2 comporte de nombreuses questions faciles. Ces dernières ne rapportent que peu de points mais vous devez néanmoins les traiter (sauter délibérément une question simple dans l'espoir d'arriver plus vite aux questions difficiles attirera l'agacement du correcteur). L'objectif est de vous entraîner à
- Détecter correctement les questions faciles.
- Les traiter efficacement : de manière concise (pas question de passer 10 lignes sur une banalité) et complète (le ou les arguments centraux doivent être présents).

Problème 1 *Séquents valides en calcul des prédicats*

Toutes les fonctions de ce problème seront implémentées en utilisant le langage Ocaml.

Dans cet exercice, on se place dans le contexte du calcul des prédicats. La première partie de l'énoncé introduit une façon de manipuler des formules du premier ordre en Ocaml en toute généralité. La seconde étudie la validité de séquents dans un langage du premier ordre fixé.

Partie 1 *Formules bien formées*

Pour manipuler des formules sur un langage du premier ordre, on introduit les types polymorphes suivants pour représenter les symboles de fonction, les symboles de relation et les termes. On introduit également un type spécifique pour les connecteurs binaires afin d'alléger l'écriture des fonctions à venir.

```
type 'f fonction = 'f*int
type 'r relation = 'r*int
type ('f,'v) terme = V of 'v | F of 'f fonction * ('f,'v) terme list
type op_binaire = Et | Ou | Implique | Equivalent
```

Les fonctions sont définies comme étant des couples : le premier élément est leur symbole, de type 'f, et le second est leur arité. La chose est similaire pour les relations. Par exemple,

```
let ex = F(('f',1),[V(4)])
```

est de type (char,int) terme et représente le terme $f(4)$. Autrement dit, les symboles de variables sont de type int et les symboles de fonction sont de type char. Attention, dans l'expression ci-dessus, 4 est bien un symbole dénotant une variable et pas une constante.

1. Déterminer le type du terme t défini ci-dessous et indiquer quel terme il représente :

```
let t = F(("plus",2), [V('x'); F(("cos",1),[V('y')])])
```

Le type définissant les termes ne peut pas vérifier que les arités des fonctions utilisées sont bien respectées, il est donc possible d'introduire un terme mal formé. Pour éviter cela :

2. Ecrire une fonction récursive terme_bien_forme de signature ('a,'b) terme -> bool renvoyant true si le terme en entrée respecte les arités des symboles de fonctions et false sinon.

On introduit à présent un type polymorphe pour manipuler des formules du premier ordre dont les variables sont de type 'v, les symboles de fonction de type 'f et ceux de relation de type 'r :

```
type ('f,'r,'v) formule =
  R of 'r relation * ('f,'v) terme list
|Forall of 'v * ('f,'r,'v) formule
|Exists of 'v * ('f,'r,'v) formule
|Non of ('f,'r,'v) formule
|Op_bin of op_binaire * ('f,'r,'v) formule * ('f,'r,'v) formule
```

3. Déclarer une variable f de type (string, string, char) formule représentant la formule

$$\forall x\, x + \cos(y) = x + \cos(y)$$

4. Comme pour les termes, le type utilisé pour les formules n'interdit pas d'écrire une formule ne respectant pas les arités de relations. Ecrire une fonction formule_bien_formee de signature ('a,'b,'c) formule -> bool indiquant si la formule en entrée est bien formée vis-à-vis de **toutes** les arités.

Dans la suite, les formules manipulées seront systématiquement syntaxiquement correctes et il ne sera pas nécessaire de vérifier que c'est le cas.

5. Ecrire une fonction apparait de signature 'a -> ('b,'a) terme -> bool telle que apparait x t indique si la variable x est présente dans le terme t.

6. Ecrire une fonction est_libre de signature 'a -> ('b,'c',a) formule -> bool telle que est_libre x f renvoie true si et seulement si une des occurrences de x est libre dans f.

7. En déduire une fonction est_close de signature ('a,'b,'c) formule -> 'c list -> bool prenant en entrée une formule f et une liste lv de variables qu'on supposera être la liste des variables intervenant dans f et indiquant si la formule f est close. Justifier brièvement sa correction.

Partie 2 *Sémantique et déduction sur un langage du premier ordre*

Dans cette partie on considère le langage du premier ordre L dont la signature est la suivante :

$$\mathcal{F} = \emptyset \text{ et } \mathcal{R} = \{=: 2, R : 2\}$$

Sur ce langage, on se donne les trois formules closes suivantes :

- $F_1 = \forall x\, R(x, x)$
- $F_2 = \forall x\, \forall y\, (R(x, y) \wedge R(y, x) \Rightarrow x = y)$
- $F_3 = \forall x\, \forall y\, \forall z\, (R(x, y) \wedge R(y, z) \Rightarrow R(x, z))$

On note $F = F_1 \wedge F_2 \wedge F_3$.

8. La structure $\mathcal{M}$ dont le domaine est $\mathbb{R}$ et dans laquelle la relation R est interprétée par l'inégalité stricte $<$ (la relation $=$ étant toujours interprétée comme étant l'égalité) est-elle un modèle pour F ? Si oui, justifier, si non, exhiber un modèle pour F.

9. Indiquer pour chacun des séquents $\Gamma \vdash G$ sur L suivants s'il est valide ou non. Si $\Gamma \vdash G$ est valide, en donner une preuve syntaxique en déduction naturelle. Si $\Gamma \vdash G$ n'est pas valide, donner une preuve sémantique de ce fait en exhibant une structure qui contredit $\Gamma \models G$.

   a) $\vdash \neg F_1 \wedge \neg F_2 \Rightarrow \neg(F_1 \vee F_2)$. Dire sans justifier si l'implication réciproque est vraie.

   b) $F \vdash \forall x \, \forall y \, (R(x,y) \vee R(y,x))$.

   c) $F \vdash \forall x \, \exists y \, R(x,y)$.

   d) $F \vdash \forall x \, \exists y \, (R(x,y) \wedge \neg(x = y))$.

   e) $F \vdash \exists x \, \forall y \, R(y,x)$

Problème 2 *Autour du mot infini de Thue-Morse*

Les définitions et notations utilisées tout au long du problème sont les suivantes :

- Un monoïde est un ensemble muni d'une loi interne, associative et disposant d'un neutre. Si $(M_1, \star)$ et $(M_2, \diamond)$ sont deux monoïdes de neutres respectifs e_1 et e_2, et f est une fonction de M_1 dans M_2, on dit que f est un morphisme de monoïdes si $f(e_1) = e_2$ et pour tous $(x,y) \in M_1^2$, on a $f(x \star y) = f(x) \diamond f(y)$.
- A désigne un alphabet à $p \geq 1$ lettres.
- On définit sur $A^\star$ la relation $\preceq$ par : $x \preceq y$ si et seulement si x est préfixe de y.
- A^∞ désigne l'ensemble $A^{\mathbb{N}}$ des "mots infinis". Il est entendu que l'appelation de "mot" pour une suite infinie de lettres est quelque peu impropre mais on l'utilisera tout de même.
- Si $u \in A^\infty$ et $n \in \mathbb{N}$, on note $[u]_n \in A^n$ le préfixe de longueur n du mot infini u.

La partie 1 est indépendante des parties suivantes.

Partie 1 *Un peu de combinatoire sur $A^\star$*

Dans cette partie, $n \in \mathbb{N}^\star$ est fixé.

1. Combien y a-t-il de mots de longueur n sur A ?
2. Parmi ces mots, combien sont formés de lettres toutes distinctes ?
3. Combien y a-t-il de palindromes de longueur n sur A ?
4. Combien A^n contient-il de mots vérifiant que deux lettres consécutives sont toujours distinctes ?
5. Si $\{a_1, ..., a_p\} \subset A$ et $n = \sum_{i=1}^{p} \alpha_i$ est une décomposition de n en une somme d'entiers naturels dont aucun n'est nul, combien y a-t-il de mots $m \in A^n$ vérifiant : $\forall i \in [\![1, p]\!], |m|_{a_i} = \alpha_i$?

Partie 2 *Un monoïde ordonné*

6. Montrer que $\preceq$ est un ordre sur $A^\star$. Est-il total ? Est-il bien fondé ?

Dans la suite, tout le vocabulaire est relatif à l'ordre $\preceq$.

7. Montrer que pour tout $u, v \in A^\star$, si $u \preceq v$ alors $|u| \leq |v|$. La réciproque est-elle vraie ?
8. L'ensemble ordonné $(A^\star, \preceq)$ possède-t-il un plus petit élément ? Un plus grand élément ?
9. Si $u, v \in A^\star$ admettent un majorant commun, montrer qu'ils sont comparables. Et sinon ?

10. Soit $(u_n)_{n\in\mathbb{N}} \in (A^\star)^{\mathbb{N}}$ une suite de mots strictement croissante.

    a) Montrer que la suite numérique $(|u_n|)_{n\in\mathbb{N}}$ est strictement croissante.

    b) Montrer qu'il existe un unique $x \in A^\infty$ tel que pour tout $n \in \mathbb{N}$, $[x]_{|u_n|} = u_n$.

L'unique élément x dont l'existence a été prouvée en question 10b s'appelle la limite de $(u_n)_{n\in\mathbb{N}}$ et on note $x = \lim\limits_{n\to+\infty} u_n$. Ainsi, les questions précédentes introduisent une notion de convergence pour les suites de mots strictement croissantes, qu'on met en application immédiatement :

11. On définit la suite $u \in (\{a,b\}^\star)^{\mathbb{N}}$ récursivement par : $\begin{cases} u_0 = ab \\ \forall n \in \mathbb{N}, u_{n+1} = u_n^2 \end{cases}$

    a) Calculer explicitement $|u_n|$ pour tout $n \in \mathbb{N}$.

    b) Montrer que u converge et calculer sa limite.

12. On définit la suite $\varphi \in (\{a,b\}^\star)^{\mathbb{N}}$ récursivement par : $\begin{cases} \varphi_0 = a \text{ et } \varphi_1 = ab \\ \forall n \in \mathbb{N},\ \varphi_{n+2} = \varphi_{n+1}\varphi_n \end{cases}$

    a) Calculer φ_2, φ_3 et φ_4.

    b) Montrer que φ converge. *Remarque : La limite de la suite φ est le mot infini de Fibonacci.*

Partie 3 *Etude du mot de Thue-Morse*

Dans cette partie, l'alphabet A est $\{a,b\}$. On y introduit quelques notations et définitions supplémentaires :

- Pour tout $m \in A^\star$, on note $\overline{m}$ le mot de $A^\star$ obtenu en remplaçant dans m toutes les lettres a par des b et toutes les lettres b par des a. Ainsi, $\overline{abbaa} = baabb$.
- Un mot de $A^\star$ est dit *sans facteur cube* si aucun de ses facteurs n'est le cube d'un mot non vide. Un mot infini $m \in A^\infty$ est dit sans facteur cube si pour tout $n \in \mathbb{N}$, $[m]_n$ est sans facteur cube.

13. Si $g : A \to A^\star$, montrer qu'il existe un unique morphisme de monoïde $f : A^\star \to A^\star$ tel que $f_{|A} = g$.

La question précédente montre que toute fonction de A dans $A^\star$ est prolongeable en un unique morphisme de monoïde de $A^\star$ dans lui même. Il fait donc sens de parler de l'unique (endo)morphisme de monoïde $\mu : A^\star \to A^\star$ tel que $\mu(a) = ab$ et $\mu(b) = ba$. On définit grâce à μ la suite $\tau \in (A^\star)^{\mathbb{N}}$ récursivement par :

$$\tau_0 = a \text{ et } \forall n \in \mathbb{N},\ \tau_{n+1} = \mu(\tau_n)$$

14. Calculer $\tau_1, \tau_2, \tau_3, \tau_4$ puis calculer explicitement $|\tau_n|$ pour tout $n \in \mathbb{N}$.

15. Montrer la stricte croissance de la suite τ.

On déduit de la question précédente l'existence d'une limite à la suite τ qu'on note $t \in A^\infty$ et dont on note les lettres $t_0, t_1, t_2...$ Le mot infini t est le *mot infini de Thue-Morse*. Dans la suite, on étudie ce mot et on en prouve une propriété remarquable.

16. Montrer que $\beta : \begin{cases} A^\star \to A^\star \\ m \mapsto \overline{m} \end{cases}$ est un morphisme de monoïdes qui commute avec μ.

17. On définit à présent deux suites mutuellement récursives x et y de mots de $A^\star$ de la façon suivante :

$$(x_0, y_0) = (a, b) \text{ et } \forall n \in \mathbb{N},\ (x_{n+1}, y_{n+1}) = (x_n y_n, y_n x_n)$$

    a) Exprimer pour tout $n \in \mathbb{N}$ les mots x_n et y_n en fonction de τ_n.

b) Pour tout $n \in \mathbb{N}$, calculer $|x_n|_a$ et $|x_n|_b$.

c) Montrer que pour tout $n \in \mathbb{N}$, x_{2n} et y_{2n} sont des palindromes.

d) Exprimer x_{2n+1} en fonction de y_{2n+1}.

18. Montrer que pour tout $n \in \mathbb{N}$, $t_n = a$ si et seulement si l'écriture binaire de n comporte un nombre pair de uns. En déduire t_{100}.

19. Montrer que si $m \in A^\star$ est sans facteur cube, alors il en va de même pour $\mu(m)$.

Indication : On pourra procéder par l'absurde, écrire $\mu(m)$ sous la forme vu^3w avec $u \neq \varepsilon$ puis discuter selon la parité de $|u|$ et $|v|$.

20. En déduire le résultat suivant du au mathématicien norvégien Axel Thue et redécouvert par Marston Morse : le mot infini t est sans facteur cube.

Partie 4 *Quelques extensions*

Dans cette partie on explore deux résultats dans des domaines très différents qui exploitent les propriétés du mot de Thue-Morse. Les questions 21, 22 sont complètement indépendantes de la question 23.

Considérons dans un premier temps la question suivante : existe-t-il un mot infini sur un alphabet de k lettres n'ayant aucun facteur non trivial répété successivement n fois ? La partie précédente assure que la réponse à cette question est oui pour $k = 2$ et $n = 3$. Qu'en est-il si $k = 2$?

21. Quels sont les mots sans facteur carré sur un alphabet $\{a, b\}$ à deux lettres ? Commenter.

22. On considère le mot infini $\theta \in \{a, b, c\}^\infty$ obtenu à partir du mot t de la façon suivante : on y remplace de gauche à droite chaque facteur abb par a, chaque facteur ab (suivi d'un b) par b et chaque facteur a (suivi d'un a) par c.

a) Montrer que si $m \in (ab + ba)^\star$, alors ama et bmb n'y appartiennent pas.

b) Montrer que t ne contient aucun facteur de la forme $avava$ ni de la forme $bvbvb$ où $v \in \{a, b\}^\star$.

c) En déduire que θ est sans facteur carré. Que vient-on de montrer ?

On considère une partie d'échecs dans laquelle le joueur blanc a deux cavaliers placés initialement en b1 et g1 (et son roi en a1 par exemple) et le joueur noir a deux cavaliers placés initialement en b8 et g8 (et son roi en a8 par exemple). Les deux séquences de coups suivants sont licites aux échecs :

- Joueur blanc déplace le cavalier de b1 en c3.
- Joueur noir déplace le cavalier de b8 en c6.
- Joueur blanc déplace le cavalier de c3 en b1.
- Joueur noir déplace le cavalier de c6 en b8.
- Joueur blanc déplace le cavalier de g1 en f3.
- Joueur noir déplace le cavalier de g8 en f6.
- Joueur blanc déplace le cavalier de f3 en g1.
- Joueur noir déplace le cavalier de f6 en g8.

23. Aux échecs, une règle stipulait : si la même séquence de coups se répète trois fois d'affilée, il est possible de rendre la partie nulle (auquel cas, elle termine). Cette règle seule suffisait-elle à éviter les parties d'échecs infinies ?

Problème 3 *Mots de Lyndon et de de Bruijn*

Toutes les fonctions de ce problème seront implémentées en utilisant le langage C.

Les définitions et notations utilisées tout au long du problème sont les suivantes :

- A désigne un alphabet. **Cet alphabet est par défaut $\{0,1\}$, y compris dans les définitions ci-dessous.** En particulier, il fait sens d'écrire $x < y$ si x et y sont des lettres de A.
- Si $u, v \in A^\star$, on dit que u est un début de v si u est un préfixe non trivial (c'est-à-dire, non vide et non égal à v) de v. De même, u est une fin de v si u est un suffixe non trivial de v.
- On définit sur $A^\star$ la relation $\prec$ de la façon suivante : si $u = u_1...u_m$ et $v = v_1...v_n$,

$$u \prec v \text{ si et seulement si } u \text{ est un début de } v \text{ ou } \exists k \in [\![1, m]\!], \begin{cases} \forall i \in [\![1, k-1]\!], u_i = v_i \\ u_k < v_k \end{cases}$$

  On définit ensuite la relation $\preceq$ par : $u \preceq v$ si et seulement si $u = v$ ou $u \prec v$. Remarquez que cette relation n'est rien d'autre que l'ordre lexicographique.
- Si $u = u_0...u_{n-1} \in A^\star$, les *conjugués* du mot u sont tous les mots $v \in A^\star$ pour lesquels il existe $i \in [\![0, n-1]\!]$ tel que $v = u_i u_{i+1}...u_{n-1}u_0...u_{i-1}$. Autrement dit, ce sont les mots obtenus à partir de u par permutation circulaire des lettres de ce mot. On dit que v est un conjugué *non trivial* de u si $i \neq 0$.
- On définit sur $A^\star$ une relation $\mathcal{C}$ de la façon suivante : u et v sont en relation via $\mathcal{C}$, ce qu'on note $\mathcal{C}(u, v)$, si et seulement si u est un conjugué de v.

Partie 1 *Préliminaires*

1. Ecrire une fonction `int inferieur(char* u, char* v)` renvoyant 0 si les chaînes u et v sont égales, une valeur strictement négative si $u \prec v$ et une valeur strictement positive si $v \prec u$.
2. Montrer que pour tout $u, v \in A^\star$, si $u \neq v$ alors ($u \prec v$ ou $v \prec u$).
3. Indiquer sans justifier si $\prec$ est compatible avec la concaténation à droite. Autrement dit, dire s'il est vrai que :
$$\forall u, v, w \in A^\star, \text{ si } u \prec v \text{ alors } uw \prec vw$$
4. Ecrire une fonction `char* conjugue(char* m, int i)` telle que `conjugue(m,i)` renvoie la chaîne de caractères correspondant au conjugué de `m` commençant par le `i`-ème caractère de `m`. On vérifiera que `i` est compris entre 0 et $|\mathsf{m}| - 1$ et on n'oubliera pas qu'une chaîne de caractère se termine par $\backslash 0$.
5. Considérons l'ensemble des conjugués d'un mot de taille n. Quelle est la taille minimale de cet ensemble ? Quelle est sa taille maximale ? Dans les deux cas, donner un exemple d'alphabet A et de mot sur A pour lequel il y a atteinte.
6. Expliquer brièvement pourquoi $\mathcal{C}$ est une relation d'équivalence sur $A^\star$.

Puisque la relation $\mathcal{C}$ est une relation d'équivalence, il fait sens de considérer des classes d'équivalences selon cette relation. Une telle classe d'équivalence s'appelle un *collier*. Par exemple, la classe d'équivalence de 001 est le collier $C_1 = \{001, 010, 100\}$ et celle de 1010 est $C_2 = \{0101, 1010\}$. On dit qu'un collier est *apériodique* si pour tout mot u dans le collier, il n'existe pas de conjugué non trivial de u qui soit égal à u. Par exemple, C_1 est apériodique mais pas C_2.

Partie 2 *Mots de Lyndon*

Un mot $u \in A^\star$ est un *mot de Lyndon* si tout conjugué non trivial v de u vérifie $u \prec v$.

7. Indiquer dans chacun des cas si le mot considéré est un mot de Lyndon. Si ce n'est pas le cas, justifier brièvement :

   a) 0010011 b) 010011 c) 001001

8. Les mots de longueur 1 sont-ils des mots de Lyndon ?

9. Ecrire une fonction `bool lyndon(char* m)` qui renvoie vrai si `m` $\neq \varepsilon$ est un mot de Lyndon et faux sinon.

La suite de cette partie explore deux façons de construire des mots de Lyndon. La première exploite le théorème suivant (qu'on ne demande pas de démontrer) : un mot est un mot de Lyndon si et seulement si c'est le plus petit élément (pour l'ordre $\preceq$) d'un collier apériodique.

10. Déterminer les classes d'équivalence selon la relation $\mathcal{C}$ (c'est-à-dire, les colliers) des mots de longueur 4 sur l'alphabet $\{0, 1\}$.

11. En déduire la liste des mots de Lyndon de longueur 4.

La seconde repose sur un algorithme permettant de générer tous les mots de Lyndon de taille bornée par un entier n. Le pseudo-code de cet algorithme est le suivant :

Entrée : Un entier $n \in \mathbb{N}$
Sortie : L'ensemble des mots de Lyndon de taille inférieure ou égale à n.

```
1. genere_lyndon(n) =
2.     m ← 0    //0 désigne bien sûr le mot constitué de la lettre 0
3.     E ← {0}
4.     Tant que m ≠ ε
5.         m ← m concaténé à lui même jusqu'à avoir un mot de taille n (la dernière occurrence
6.         de m sera éventuellement tronquée pour arriver à un mot de taille exactement n)
7.         Tant que la dernière lettre de m vaut 1
8.             m ← m privé de sa dernière lettre
9.         Si m ≠ ε, remplacer la dernière lettre de m par 1
10.        E ← E ∪ {m}
11.    Renvoyer E
```

12. Si le mot m qui entre dans la boucle tant que de la ligne 4 est le mot 00111 et que $n = 9$, indiquer quel est le mot qui sera ajouté à l'ensemble E suite à l'exécution des lignes 4 à 9.

13. Construire sans justifier l'ensemble des mots de Lyndon de taille au plus 4. On écrira ces mots dans l'ordre dans lequel l'algorithme les produit.

14. Justifier la terminaison de cet algorithme.

15. Majorer grossièrement sa complexité pire cas en considérant qu'ajouter ou supprimer une lettre à un mot se fait en temps constant.

Partie 3 *Factorisations de Lyndon*

Si $u \in A^\star$, on appelle *factorisation de Lyndon* de u toute suite $(u^{(1)}, u^{(2)}, ..., u^{(n)})$ de mots de Lyndon telle que $u^{(1)} \succeq u^{(2)} \succeq ... \succeq u^{(n)}$ et $u = u^{(1)}u^{(2)}...u^{(n)}$. Autrement dit, une factorisation de Lyndon de u consiste en un découpage de u en mots de Lyndon de plus en plus petits. Par exemple, le mot $m_{ex} = 011100010$ admet comme factorisation de Lyndon :

$$m_{ex} = (0111)(0001)(0)$$

On admet le théorème suivant : tout mot $u \in A^\star$ admet une factorisation de Lyndon. On cherche dans les questions suivantes à construire algorithmiquement une telle factorisation. Pour ce faire, on considère l'algorithme $\mathcal{A}$ suivant. On souhaite factoriser le mot u.

- Initialement on factorise $u = u_1u_2...u_n$ de taille n en $u^{(1)}, ..., u^{(n)}$ ou pour tout $i \in [\![1, n]\!]$, $u^{(i)}$ est la i-ème lettre de u, à savoir u_i. Autrement dit, initialement chaque facteur est une lettre.
- Si à une étape de l'algorithme on a obtenu une factorisation $u = u^{(1)}u^{(2)}...u^{(p)}$ en mots de Lyndon et qu'il existe $k \in [\![1, p-1]\!]$ tel que $u^{(k)} \prec u^{(k+1)}$, on modifie la factorisation obtenue en une nouvelle factorisation plus courte $v^{(1)}v^{(2)}...v^{(p-1)}$ définie par

$$\begin{cases} v^{(j)} = u^{(j)} & \text{pour tout } j \in [\![1, k-1]\!] \\ v^{(k)} = u^{(k)}u^{(k+1)} & \\ v^{(j)} = u^{(j+1)} & \text{pour tout } j \in [\![k+1, p-1]\!] \end{cases}$$

16. En utilisant l'algorithme décrit ci-dessus, donner une factorisation de Lyndon de 10100101100100.
17. Montrer que l'algorithme $\mathcal{A}$ fournit effectivement une factorisation de Lyndon de son entrée en admettant le résultat suivant : si u et v sont deux mots de Lyndon tels que $u \prec v$ alors uv reste un mot de Lyndon.

Pour implémenter cet algorithme en C, on choisit de représenter la factorisation d'un mot par un tableau de booléens avec les conventions suivantes : la case i contient `true` si et seulement si la i-ème lettre de u est la fin d'un des facteurs. Par exemple, la factorisation du mot m_{ex} sera représentée par le tableau

false	false	false	true	false	false	false	true	true

18. Ecrire une fonction `char* extrait_chaine(char* mot, int debut, int fin)` qui extrait de la chaîne `mot` la chaîne comprise entre les caractères indicés par `debut` et `fin` (inclus). On rappelle qu'une chaîne doit finir par le caractère `\0`.
19. Ecrire une fonction `bool est_trie(char* mot, bool* facto, int* defaut_de_tri)` prenant en entrée un mot non vide, une factorisation de ce mot avec les conventions ci-dessus et l'adresse d'un entier. Si la factorisation proposée est une suite décroissante de mots, elle renverra `true`. Sinon, elle renverra `false` et stockera l'indice de fin d'un des mots strictement plus petit que son successeur dans `defaut_de_tri`. On expliquera clairement le fonctionnement de la fonction proposée.
20. En déduire une fonction `bool* factorisation(char* mot)` qui calcule une factorisation de Lyndon de son entrée.

Partie 4 *Mots de de Bruijn*

Dans cette partie, A désigne un alphabet quelconque. Un *mot de de Bruijn* d'ordre $n \in \mathbb{N}^\star$ est un mot de $A^\star$ qui contient tous les mots de A^n une et une seule fois lorsqu'on le considère circulairement. Par exemple *abacbbcca* est un mot de de Bruijn d'ordre 2 sur $\{a, b, c\}$ car il contient une et une seule fois chaque mot de longueur 2 de $\{a, b, c\}$, le mot *aa* étant obtenu en considérant que la lettre suivant le a final est la première lettre du mot, ici a.

21. Le mot *abba* est-il un mot de de Bruijn sur $\{a, b\}$? Si oui, quel est son ordre ?

22. Quelle est la longueur d'un mot de de Bruijn en fonction de son ordre n et du nombre k de lettres dans A ?

Les questions suivantes explorent deux méthodes permettant de construire des mots de de Bruijn. La première repose sur la notion de *graphe de de Bruijn*. Le graphe de de Bruijn d'ordre n sur un alphabet A à k lettres est un graphe étiqueté oriente $B(k, n)$ tel que :

- Les sommets de $B(k, n)$ sont étiquetés par les mots de A^n.
- Les arêtes sont données par l'ensemble $\{(am, mb, b) \mid a, b \in A \text{ et } m \in A^\star\}$. Par exemple $(001, 010, 0)$ est une arête dans $B(2, 3)$ lorsque l'alphabet est $\{0, 1\}$: elle va du sommet 001 vers le sommet 010 et est étiquetée par 0.

23. Dessiner le graphe de de Bruijn $B(2, 3)$ lorsque l'alphabet est $\{0, 1\}$.

24. Montrer que le degré sortant et le degré entrant de chaque sommet dans $B(k, n)$ est égal à k.

25. En déduire le nombre d'arcs dans le graphe $B(k, n)$.

Un circuit eulérien dans un graphe orienté G est un circuit passant une et une seule fois par chaque arc de G. On admet le résultat suivant : un graphe orienté G admet un circuit eulérien si et seulement si il est connexe et pour tout sommet s de G, les degrés entrant et sortant de s sont égaux.

26. Construire $B(2, 2)$ et déterminer un circuit eulérien dans ce graphe. Constater que le mot obtenu par concaténation des étiquettes des arêtes parcourues lors de ce circuit est un mot de de Bruijn. Quel est son ordre ?

Ce résultat est en fait général : un circuit eulérien dans le graphe $B(k, n)$ produit un mot de de Bruijn d'ordre $n + 1$ sur k symboles. En admettant ce fait :

27. Montrer que pour tout alphabet A, et tout $n \geq 2$, il existe un mot de de Bruijn sur l'alphabet A d'ordre n.

Une autre façon de construire un mot de de Bruijn est de construire... des mots de Lyndon, dont la génération a été étudiée en partie 2. Si $n \in \mathbb{N}$, concaténer dans l'ordre lexicographique tous les mots de Lyndon sur A dont la longueur divise n produit un mot de de Bruijn d'ordre n sur A. Le mot de de Bruijn obtenu est même le plus petit parmi tous les mots de de Bruijn d'ordre n sur A.

28. Calculer le plus petit mot de de Bruijn d'ordre 4 sur $\{0, 1\}$.

L'accès à votre immeuble est protégé par un digicode. Pour entrer, il faut taper une séquence de n chiffres entre 0 et $k - 1$. Le digicode fonctionne de la façon suivante : à chaque nouveau chiffre entré après le n-ème, il vérifie si les n derniers chiffres correspondent au code d'entrée. Par exemple, si $n = 4$ et qu'on tape 021201, le digicode teste successivement les codes 0212, 2120 puis 1201.

29. Combien de codes faudrait-il a priori tester pour pouvoir entrer dans l'immeuble ? Si on les essaie tous successivement, combien de chiffres faudrait-il taper dans le digicode ?

30. Vu le fonctionnement du digicode, proposer une séquence astucieuse de chiffres qui garantit l'ouverture de la porte tout en diminuant le nombre total de chiffres (qu'on précisera) à entrer dans le digicode. Expliquer brièvement votre raisonnement.